A PARALLEL SIMULATED ANNEALING ALGORITHM FOR STANDARD CELL PLACEMENT ON A HYPERCUBE COMPUTER

Prithviraj Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Av.
Urbana, IL-61801

(217) 333-6564

ABSTRACT

A parallel processing algorithm for standard cell placement suitable for execution on a Hypercube computer is presented. In the past, several parallel algorithms for performing module placement have been proposed that are suitable for execution on a two-dimensional array of processors. Those algorithms had several limitations, namely, they got stuck at local minima, were susceptible to oscillations, could not handle variable sized modules (standard cells), and allowed only nearest neighbor exchanges. Recently, the simulated annealing technique has been applied to solve the standard cell placement problem on conventional uniprocessor computers. These algorithms do not get stuck at local minima and can handle modules of variable sizes, but take an extremely long time to be executed. In this paper, a parallel version of the simulated annealing technique is presented which is targeted to run on a Hypercube computer such as the Intel iPSC. We discuss how the cells in a two-dimensional area of a chip are mapped onto processors in an n-dimensional hypercube such that both small and large moves can be applied. Two types of moves are allowed: cell exchanges and cell displacements. Initially, at high temperatures of the annealing process, moves are allowed in all dimensions of the hypercube. As the temperature is decreased, certain dimensions of the hypercube are selectively "frozen" to restrict the range of the moves. The computation of the cost function in parallel among all the processors in the hypercube is described along with a distributed data structure that needs to be stored in the hypercube to support such a parallel cost evaluation. The existence of Hamiltonian circuits in the hypercube topology is utilized to broadcast the results of cell moves to all processors. Initial estimates show that the algorithm is several orders of magnitude faster than existing simulated annealing-based algorithms running on conventional uniprocessors. The algorithm also does not have any of the above-mentioned limitations of iterative-improvement placement algorithms for two-dimensional processor arrays.

Acknowledgment: This research was supported by the National Aeronautics and Space Administration under Contract NASA NAG 1-613.

I. INTRODUCTION

Given a set of standard cells of constant height and variable width, and a net list which describes the interconnections among the cells, our objective is to place the cells so as to minimize the total length of wires interconnecting the cells. Conventional module placement algorithms [1, 2], that were designed to run sequentially on uniprocessor computers, take a long time to solve. Hence, several algorithms have been proposed recently to speed up the problem of module placement by implementing them on two dimensional processor arrays [3, 4]. However, these array algorithms have several limitations, namely, they get stuck at local minima, are susceptible to oscillations, can perform nearest neighbor exchanges only, and cannot handle variable-sized modules (standard cells).

Recently, the simulated annealing technique [5] has been applied to the placement problem in a program called TimberWolf which, by applying cell displacements and exchanges randomly, avoids getting stuck at local minima and thereby achieves near-optimal placement [6]. A major limitation of TimberWolf is that it is extremely slow. In this paper, we present a parallel implementation of a simulated annealing algorithm for cell placement on a Hypercube computer. The parallel algorithm does not have any of the limitations of the array processor algorithms mentioned above and is several orders of magnitude faster than TimberWolf running on a VAX-11/780.

A hypercube topology consists of 2^d processors that are interconnected through the topology of a cube in d dimensions. A 4-dimensional hypercube is shown in Fig. 1. Several prototypes of such machines have been built [7, 8]; one of them is now available commercially from Intel [9].

II. PARALLEL ALGORITHM FOR CELL PLACEMENT ON A HYPERCUBE

We now describe an algorithm for performing the standard cell placement using a variation of the TimberWolf [6] algorithm on a hypercube of 6 dimensions connecting 64 processors. The algorithm can be easily generalized to hypercubes of other dimensions. Let us suppose that we are given the problem of placing N standard cells where $N >> 64$.

Cell Assignment to Processors

We now describe how cells in a two-dimensional area of a chip are assigned to processors that are connected in the form of a 6-dimensional hypercube. For the hypercube topology, a processor whose binary address is $p_5p_4 \cdots p_i \cdots p_0$ is connected to processor $p_5p_4 \cdots \overline{p_i} \cdots p_0$ via a link in dimension i. We propose that each processor be assigned an approximately equal area portion of the total chip area which can be viewed as a virtual 8×8 square grid. We initially assign the cells to different processors such that the sums of areas of cells assigned to each processor is approximately equal to $A_{average} = \frac{1}{64}\sum_{m=1}^{N} A_m$, where A_m is the area of the m^{th} cell. The cells within each processor are initially placed in a single row with no area overlap between them. Since all cells have constant height, each processor therefore is assigned a rectangular portion of the chip area. The correspondence between processor addresses and grid points on the physical chip area is shown in Fig. 2, by choosing which we guarantee that the processors that are adjacent in a predetermined set of four dimensions of the hypercube allow all nearest North-South-East-West neighbor exchanges. The other two dimensions of the hypercube are used for exchanges across larger distances in the area map. For example in Fig. 2, processor 26, which controls grid location (3,4), has a 4-link to processor 10, 3-link to processor 18, 2-link to processor 30, and 0-link to processor 27, which correspond to the nearest neighbors in the North(2,4), South(4,4), East(3,5) and West(3,3) directions; in addition, the 1-link to processor 24 and the 5-link to processor 58, control grid locations, (3,1) and (6,4), that are distance 3 away from (3,4).

Moves

For each temperature of the simulated annealing process, we define two types of moves: cell exchange and cell displacement. At high temperatures during the simulated annealing process, we allow exchanges and displacements of cells in all dimensions, i.e., small and large changes. Gradually, as the temperature is decreased, for each processor, certain dimensions are "frozen", i.e. changes between pairs of processors connected via those dimensions are inhibited. The temperature

intervals at which various physical distance changes are to be inhibited are explicitly specified by the user as a table of temperature intervals versus distance intervals. The parallel algorithm for cell placement is outlined in Fig. 3.

Distributed Data Structure

We assume that each processor contains the following information to aid the computation of the cost function in parallel among processors in the hypercube: (1) A list of currently assigned cells along with the following information for each cell: (2) The (x,y) coordinate location at which the center of the cell is currently placed; (3) The width of the cell; (4) A list of nets to which this cell is connected to; (5) For each net listed in (4), a list of other cells to which the net is connected, along with their (x,y) locations; (6) A list of all (x,y) locations and widths of all cells that are assigned to processors that are adjacent in two dimensions of the hypercube corresponding to the East-West nearest neighbors in the physical area map.

Parallel Calculation of Cost Function

Let us consider the exchange class move first. We denote the two processors participating in the move to be P and Q, and the respective cells that are to be considered for exchange by CELL(P) and CELL(Q). The resultant change in cost function consists of four terms.

$$\Delta_{exchange} = \Delta_1(CELL(P),P) + \Delta_2(CELL(Q),Q) + \Delta_3(CELL(P),Q) + \Delta_4(CELL(Q),P)$$

The term $\Delta_1(CELL(P),P)$ deals with the change in the wire length due to the movement of CELL(P) from $(x,y)_{CELL(P)}$ to the $(x,y)_{CELL(Q)}$, and is calculated by estimating the change in half the perimeter of the bounding box of each net. This term can be calculated by processor P alone since it keeps information about all the nets to which CELL(P) is connected, along with all the (x,y) locations of cells that are on the same nets, and can read the (x,y) location of CELL(Q) (which is the new (x,y) location for CELL(P)) from processor Q. We assume that the nets are connected to the center of each cell. The term $\Delta_2(CELL(Q),Q)$ relates to the change in wire length due to the movement of CELL(Q) from $(x,y)_{CELL(Q)}$ to $(x,y)_{CELL(P)}$, and is computed in an ident-

ical manner in parallel by processor Q.

The term $\Delta_3(CELL(P),Q)$ deals with the change in the area overlap due to the movement of CELL(P) from $(x,y)_{CELL(P)}$ to $(x,y)_{CELL(Q)}$, and is calculated by processor Q since it has information about all the cells that are near a given (x,y) location within processor Q's area map. If CELL(P) is placed somewhere within the processor Q's area map, the cell may overlap with any cell in processor Q, or with cells in the processors that are adjacent to processor Q in the two dimensions of the hypercube that correspond to its East-West neighbors in the physical area map. The term $\Delta_4(CELL(Q),P)$ deals with the change in the area overlap due to the movement of CELL(Q) from $(x,y)_{CELL(Q)}$ to $(x,y)_P$, and is performed in an identical manner in parallel by processor P.

Finally, we outline how the cost function is calculated for a displacement class move. We assume that the processors that take place in the displacement are P and Q, and the cell that is being displaced is CELL(P). Then the resultant change in cost function is

$$\Delta_{displace} = \Delta_1(CELL(P),P) + \Delta_2(CELL(P),P) + \Delta_3(CELL(P),Q)$$

where $\Delta_1(CELL(P),P)$, computed by processor P, is the change in wire length due to the movement of CELL(P) from $(x,y)_{start}$ to $(x,y)_{final}$; $\Delta_2(CELL(P),P)$, computed by processor P, is the change in area overlap caused by the movement of CELL(P) from $(x,y)_{start}$ to $(x,y)_{final}$, if $(x,y)_{final}$ lies in processor P's area map, otherwise the term is zero; $\Delta_3(CELL(P),Q)$, computed by processor Q, is the change in area overlap caused by the movement of CELL(P) from $(x,y)_{start}$ to $(x,y)_{final}$ if $(x,y)_{final}$ lies in processor Q's area map, otherwise the term is zero.

Broadcasting New Cell Locations

Once the cells have been moved, the updated cell locations have to be sent to all processors. The scheme used to perform this broadcast uses the property of the existence of Hamiltonian circuits in the hypercube topology. A Hamiltonian circuit in a graph is a closed walk that traverses every vertex of the graph exactly once, except the starting vertex at which the walk also terminates [10]. A Hamiltonian circuit for the 4-dimensional hypercube is highlighted in Fig. 1. Each

processor which has an updated cell location will inform its Hamiltonian circuit successor of the updated value of the cell location. It can be easily seen that if all 64 processors contained updated cell locations, it will take at most 64 time steps for all the updated cell locations to be available at all the processors.

III. ALGORITHM PERFORMANCE

The advantage of our algorithm over TimberWolf is that it is much faster. We are currently implementing the algorithm in the C programming language using a Hypercube simulator running on a Sequint computer under UNIX. The results of the simulation will be reported at the conference.

We give here an estimate of performance of the proposed algorithm through an example. In TimberWolf, the number of new states that are generated for each temperature in the annealing process is about 100 times the number of standard cells. For a 1000 cell circuit, the number of new states generated per stage (temperature) is 100,000. For 100 stages, TimberWolf would take about 12 hours of CPU time on a VAX-11/780 running VMS, assuming that it takes 4 milli-seconds to generate and evaluate each new state [6]. If the placement is performed using our algorithm on a hypercube of six dimensions, we postulate that approximately 25% extra number of moves will be required for convergence per stage because the hypercube algorithm allows only restricted moves compared to unrestricted moves in TimberWolf. Hence, the number of hypercube steps required to generate 125,000 new states per stage is about 4000 since the moves can be performed in parallel on 64 processors (32 moves at a time). The VAX-11/780 is reported to operate at 0.5 MIPS (millions of instructions per second) [11], and the Intel iPSC Hypercube is reported to be 0.8 MIPS per processing node [9]. Hence, the time taken for computation for each parallel move should be about 3 milli-seconds on each processing node. We assume that each broadcasting of a change in cell location requires 500 bytes (parameters that are needed to be passed include the cell's new location, list of nets, list of cells on nets and their locations, sizes, etc.). The time for one message transfer per link is 0.1 milli-seconds assuming a 10 MBytes/second hypercube link [9]. The broadcast can be

performed in 64 time steps, hence the time for communication is 3.2 milli-seconds. The total time per stage for one set of parallel moves and broadcast of its result to all processors takes about 10 milli-seconds including some time for synchronization. Hence it should take about one hour for performing the standard cell placement for 1000 cells in 100 stages on the hypercube. We therefore expect to get about an order of magnitude speed improvement with our placement algorithm for 1000 cell placement using a 64 node hypercube. We hope to achieve several orders of magnitude speed improvement in a 1024 node hypercube that is being designed at Caltech under the MARK-III project [7].

IV. CONCLUSIONS

We note the following points about our parallel algorithm.

(1) Unlike the array algorithms for module placement[3, 4] the proposed algorithm will not get stuck at local minima since we are applying the simulated annealing technique.

(2) Our algorithm will not give rise to oscillations because we have a number of cells assigned to each processor. A cell is chosen randomly for possible exchange with another randomly chosen cell in an adjacent processor. Hence, even though the cell exchanges are performed in parallel based on (x,y) locations of cells from a previous iteration step, the possibilities of choosing the same pair of cells for repeated exchange (oscillations) is very low.

(3) Cell exchanges can be performed among nearest neighbors through our novel area mapping technique and also between cells that are large distances away. Hence, movement between states that are both near and far apart in the solution space are allowed.

(4) The algorithm can handle cells with unequal areas.

(5) Our algorithm is at least an order of magnitude faster than TimberWolf running on a VAX-11/780.

REFERENCES

[1] M. Hanan and J. M. Kurtzberg, "Placement Techniques," in *Design Automation of Digital Systems: Theory and Techniques*, ed., M. A. Breuer. Prentice-Hall, pp. 213-282, 1972.

[2] M. A. Breuer, "Min-cut Placement," *Jour. Design Automation and Fault Tolerant Computing*, vol. 1, pp. 343-382, Oct. 1977.

[3] D. J. Chyan and M. A. Breuer, "A Placement Algorithm for Array Processors," *Proc. 20th Design Automation Conf.*, pp. 182-188, Jun. 1983.

[4] K. Ueda, T. Komatsubara, and T. Hosaka, "A Parallel Module Placement Approach for Logic Module Placement," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*. pp. 39-47, Jan. 1983.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, May 1983.

[6] C. Sechen and A. S. Vincentelli, "The TimberWolf Placement and Routing Package," *Proc. Custom Integrated Circuits Conf.*, pp. 522-527, May 1984.

[7] J. Tuazon, J. Peterson, M. Pniel, and D. Leberman, "Caltech/JPL Mark II Hybercube Concurrent Processor," *Proc. 1985 Parallel Processing Conference*, pp. 666-673, Aug. 1985.

[8] J. C. Peterson, J. Tuazon, D. Lieberman, and M. Pniel, "The Mark III Hypercube-Ensemble Concurrent Computer," *Proc. 1985 Parallel Processing Conference*, pp. 71-73, Aug. 1985.

[9] Intel Scientific Computers, "iPSC: The First Family of Concurrent Supercomputers," 1985, product announcement.

[10] N. Deo, in *Graph Theory with Applications to Engineering and Computer Science*. Englewoods Cliffs, N.J.: Prentice-Hall, Inc., 1974 .

[11] J. S. Emer and D. W. Clark, "A Characterization of Processor Performance in the VAX-11/780," *Proc. 11th Int. Symp. on Computer Architecture*, pp. 301-310, Jun. 1984.

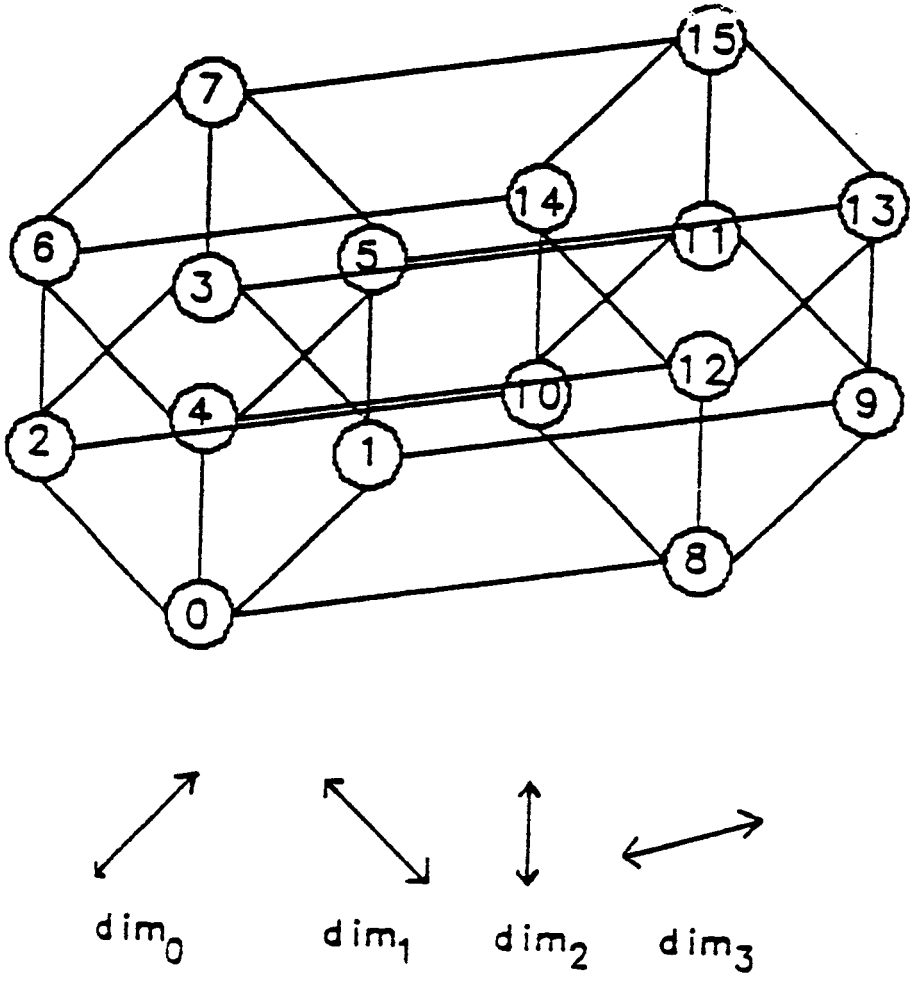

Fig. 1. A 4-dimensional hypercube topology.

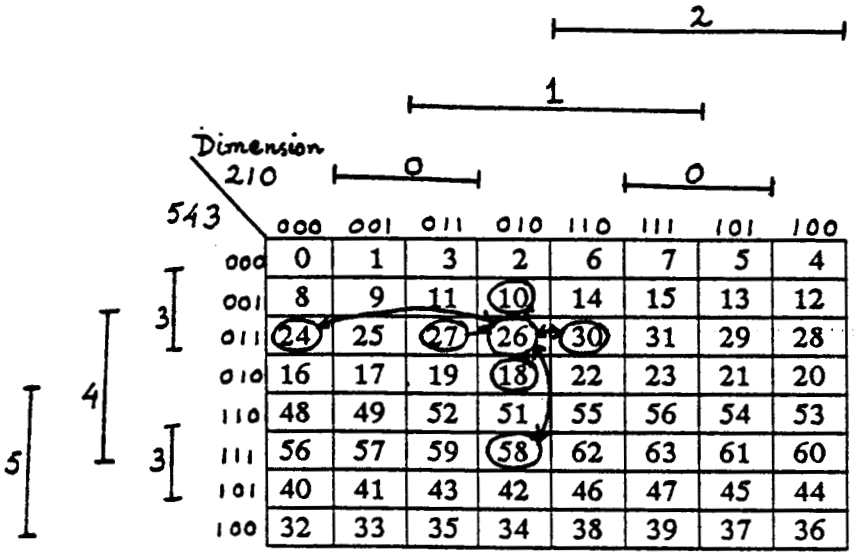

Fig. 2. Area map of 64 processor hypercube.

```
PROCEDURE placement:
   Choose initial placement using equal area criterion.
   Estimate the initial total wire length for interconnecting all
   cells by assuming wire length of each net to be half the perimeter
   of the bounding box for the net.
   maxdist := 7; (* maximum grid distance of any move *)
   WHILE "stopping criteria" is not satisfied DO BEGIN
      Generate T' < T ; (* user specified *)
      IF T_{m-1} < T' < T_m
         THEN maxdist := m ; (* user specified freezing of dimensions *)
      FOR d := 0 to 5 DO BEGIN
         FOR each processor PARALLEL DO BEGIN
            Select a cell randomly; Denote cell chosen in processor p by CELL (p );
         ENDFOR;
         FOR EACH adjacent processor pair (p ,q ) in dimension d PARALLEL DO BEGIN
            Assign p to be Master Processor, q to be Slave Processor randomly;
            IF physical distance (p ,q ) ≤ maxdist THEN
               Perform cell-exchange computations regarding CELL (p ) and CELL (q );
               Perform cell-exchange decision regarding CELL (p ) and CELL (q )
               using probabilistic ACCEPT (c ,c' ,T' ) function on Master Processor;
            ENDIF;
         ENDFOR;
         FOR EACH successful exchange PARALELL DO BEGIN
            Broadcast new cell locations from Master Processors to all
            processors using Hamiltonian circuit concept;
         ENDFOR;
         FOR each processor pair (p ,q ) connected in dimension d PARALLEL DO BEGIN
            Assign p to be Master and q to be Slave randomly;
            Select a cell randomly in Master; Denote cell by CELL (p );
            Select a random x -location within area map of either Master
            or Slave processor to which to displace cell CELL (p );
            Perform decision using probabilistic ACCEPT (c ,c' ,T' )
            function on Master;
         ENDFOR;
         FOR each successful displacement PARALLEL DO BEGIN
            Broadcast successful new cell locations from all Master processors
            to all other processors using Hamiltonian circuit concept;
         ENDFOR;
      ENDFOR;
   ENDWHILE;
END;
```

Fig. 3. Parallel algorithm for cell placement.